Notes on using Analog Devices' DSP, audio, & video components from the Computer Products Division
Phone: (800) ANALOG-D or (781) 461-3881, FAX: (781) 461-3010, EMAIL: dsp_applications@analog.com

Using Release 3.2 Of The ADSP-21000 Development Software To Generate ADSP-21061 Executables.

Last Modified: 11/14/96

Introduction

This Note describes how to develop programs for the ADSP-21061 using Release 3.2 of the ADSP-21000 Family Software Development Tools.

Because the ADSP-21061 became available after the 3.2 release of the software tools, these tools do not have built-in support for the ADSP-21061. As a result, keywords of the type "ADSP21061" cause an error if used with the 3.2 release assembler, linker, or compiler. The next release of the software tools, which is scheduled for January 1997, will have built-in support for the ADSP-21061.

Because the ADSP-21061 is object code compatible with the ADSP-21062, the techniques required for developing ADSP-21061 code with the 3.2 release tools involve accommodating the functional differences between the ADSP-21061 and ADSP-21062. These techniques involve the following development topics:

- Use the development tools architecture (.ACH) file to accommodate memory structure differences between the ADSP-21061 and ADSP-21062.
- Avoid using features of the ADSP-21062 that are not on the ADSP-21061.
- Be careful when using features of the ADSP-21062 that are different on the ADSP-21061.

Producing ADSP-21061 Code

You can use release 3.2 software tools to generate code for the ADSP-21061. Because the ADSP-21061 is object code compatible with the ADSP-21062, you can produce code that the tools recognize as ADSP-21062 code and run that code on an ADSP-21061.

The technique is to use the ADSP21062 keyword as an architecture or assembly file directive and run the output code on your ADSP-21061 system. Developing code this way only works as long as the you keep in mind the functional differences of the two DSPs and accommodate these differences within your code and system architecture file. As described in the ADSP-21061 Data Sheet, the ADSP-21061 differs from the ADSP-21062 in the following ways:

- The ADSP-21061 has one Megabit on -chip SRAM that is organized in two blocks with eight columns that are 4K deep.
- The ADSP-21061 does not have Link ports.
- The ADSP-21061's handshakes for external port DMA pins DMAR2 and DMAG2 are assigned to external port DMA channel 6.
- The ADSP-210601 does not have bi-directional SPORT DMA.
- The ADSP-21061 does not have DMA channels 8 and 9.
- The ADSP-21061's modify registers in SPORT DMA are not programmable.

ADSP-21061 Internal Memory

To develop ADSP-21062 code and run it on an ADSP-21061, you must be aware of (and your code must accommodate) how the memory addressing reflects the physical memory configuration. The correspondence between the internal memory organization the ADSP-21061 and ADSP-21062 appears in Figure 1.

From Figure 1, one should note that any program or data word placed in the upper half of Block 0 of a ADSP-21062's memory would also be placed in memory Block 0 if the program was run on an ADSP-21061. One should also note, as shown in Figure 1, that any contents placed in memory Block 1 of an ADSP-21062 would also be placed in Block 1 of an ADSP-21061.

The ADSP-21062 has 2 Mbits of SRAM is organized in 2 blocks. Each block has 16 columns and each column is 16-bits wide with a height of 4K.

The ADSP-21062 has 1 Mbit of SRAM organized in 2 Blocks. Each block is organized as 8 columns,

each 16-bits wide with a height of 4K. Figure 2 shows the bank and column structure of the ADSP-21061's internal memory.

To write ADSP-21061 compatible code, you must ensure that the architecture file does not specify more than the eight 16-bit columns per RAM block that are on the ADSP-21061. Your architecture file must handle this issue because the 3.2 release linker does not check for this type of error.

Note: The work arounds described here do not work for the 3.2 release EZ-ICE emulator. There is no support for the ADSP-21061 DSP using the 3.2 release EZ-ICE emulator. The next release of the software tools, which is scheduled for January 1997, will have emulator support for the ADSP-21061.

Notes on the example

You can run the example, which appears at the end of this note, on an ADSP-21061 or an ADSP-21062 system. The architecture file, exam61.ach, shows how you can use the 3.2 release tools to program for the ADSP-21061's memory configuration. You can use the example assembly file, exam61.asm, as a test case for the simulator or a DSP system.

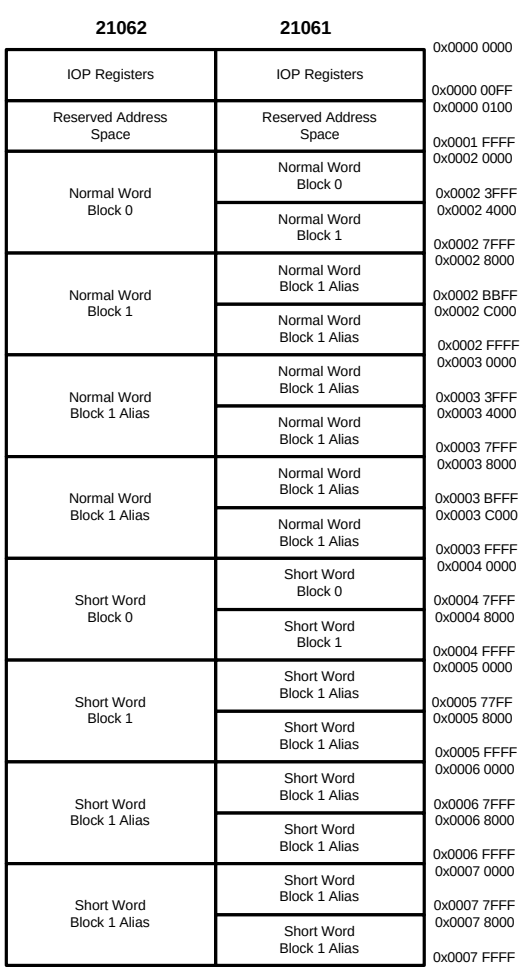

Figure 1 ADSP-21061 & ADSP-21062 Memory

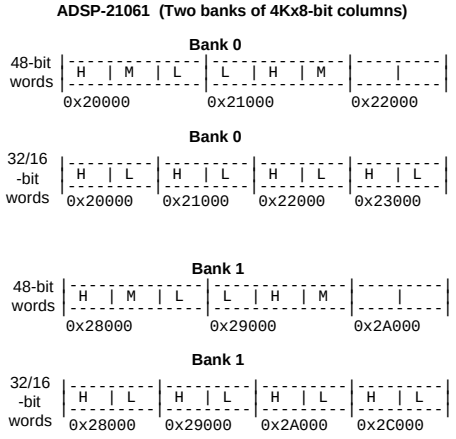

Figure 2 ADSP-21061 Memory Structure

Listing 1 - exam61.ach

```
/* This example architecture file can be used with the ADSP-21062 or the ADSP-21061 because the      aliased area of
block 1 starting at 0x28000 is used (0x28000 aliases to 0x24000 on the ADSP-21061). The      aliased area is used because
the Release 3.2 SHARC linker does not understand the ADSP-21061 memory map. The user must ensure the architecture
file does not specify more than the 8 16-bit columns per RAM block on the ADSP-21061 because the Release 3.2 linker
will not check for this. */

.SYSTEM         EZ_LAB;
.PROCESSOR =    ADSP21062;

.SEGMENT /RAM  /BEGIN=0x020000 /END=0x0200FF /PM /width=48       isr_tabl; ! 8K x 48 in block 0
.SEGMENT /RAM  /BEGIN=0x020100 /END=0x021FFF /PM /width=48    code0;
.SEGMENT /RAM  /BEGIN=0x023000 /END=0x023FFF /DM /width=32    data0;    ! 4k x 32 in block 0
.SEGMENT /RAM  /BEGIN=0x028000 /END=0x028FFF /PM /width=48    code1;    ! 4K x 48 in block 1
.SEGMENT /RAM  /BEGIN=0x02a000 /END=0x02bFFF /DM /width=32    data1;    ! 8K x 48 in block 1
.ENDSYS;
```

Listing 2 - exam61.asm

```
/* Test of instruction and data access of both internal RAM blocks */
.SEGMENT/DM       data0;
.VAR     testdata0[2]=1.0,2.0;
.ENDSEG;

.SEGMENT/DM       data1;
.VAR     testdata1[2]=3.0,4.0;
.ENDSEG;

.SEGMENT/PM       isr_tabl;
         nop;nop;nop;nop;
         nop;jump block0;nop;nop;
.ENDSEG;

.SEGMENT/PM       code0;
block0:  f0=dm(testdata1); /* should take 1 cycle */
         f1=dm(testdata1+1);
         f2=f0+f1;

/* These take 2 cycles due to instruction/data conflict in block */
         f0=dm(testdata0);
         f1=dm(testdata0+1);
         f3=f0+f1;

/* These take 2 cycles the first time through and 1 cycle subsequent accesses due to the      cache*/
         f0=pm(testdata0);
         f1=pm(testdata0+1);
         f4=f0+f1;

         jump block1;
.ENDSEG;

.SEGMENT/PM       code1;
block1:  f0=dm(testdata0); /* should take 1 cycle */
         f1=dm(testdata0+1);
         f2=f0+f1;

/* These take 2 cycles due to instruction/data conflict in block */
         f0=dm(testdata1);
         f1=dm(testdata1+1);
         f3=f0+f1;

/* These take 2 cycles the first time through and 1 cycle subsequent accesses due to the      cache*/
         f0=pm(testdata1);
         f1=pm(testdata1+1);
         f4=f0+f1;

         jump block0;
.ENDSEG;
```